Dokumentacja techniczna

Wieloboki Voronoi

Sylwia Marek
Ryszard Pręcikowski
wtorek B, 14:40

2 stycznia 2021

1 Wprowadzenie

Projekt zawiera porównanie metod konstrukcji wieloboków Voronoi dla zadanego zbioru punktów na płaszczyźnie, za pomocą następujących algorytmów:

1.1 Opis problemu

Diagram Voronoi – podział płaszczyzny, który dla danego zbioru n punktów generuje n obszarów, w taki sposób, że losowy punkt trafiający do danego obszaru znajduje się bliżej danego punktu (ze zbioru n punktów), niż od pozostałych ($n - 1$ punktów). Do wyznaczania odległości została użyta metryka Euklidesowa.

1.2 Algorytm Fortune'a

Za pomocą algorytmu zamiatania tworzy diagram Voronoi dla zbioru punktów na płaszczyźnie.

1.3 Algorytm Bowyera-Watsona (triangulacja Delaunay'a)

Triangulacja Delaunay'a jest grafem dualnym diagramu Voronoi. Na jej podstawie dla zadanej chmury punktów możliwe jest wyznaczenie diagramu Voronoi poprzez połączenie odcinkami środków okręgów opisanych na sąsiadujących trójkątach oraz utworzenie odpowiednich półprostych.

2 Wymagane pakiety

2.1 Wizualizacja

1. numpy
2. matplotlib
3. json

2.2 Główne algorytmy

1. numpy
2. matplotlib
3. math
4. random

5. typing
6. dataTypes
7. computing
8. queue
9. time

3 Źródła

1. M. de Berg, O. Cheong, M. van Kreveld, M. Overmars, *Computational Geometry: Algorithms and Applications 3rd Edition*
2. https://en.wikipedia.org/wiki/Circumscribed_circle#Circumscribed_circles_of_triangles
3. https://en.wikipedia.org/wiki/Delaunay_triangulation
4. https://math.stackexchange.com/questions/213658/get-the-equation-of-a-circle-when-given-3-points?fbclid=IwAR2THXafz_TgKEfPpPuq4YnHrANNj17lxbkVWtMs0emioIWgW6_9q8Dzt4c
5. http://www.multimedia.edu.pl/for_students/teaching_resources/biometry/files/cwiczenie6b.pdf
6. https://pvigier.github.io/2018/11/18/fortune-algorithm-details.html
7. https://stackoverflow.com/questions/85275/how-do-i-derive-a-voronoi-diagram-given-its-point-set-and-it

4 Opis programu/algorytmów

Do wizualizacji algorytmów wykorzystano narzędzea do rysowania konstrukcji geometrycznych wykorzystywanych w trakcie laboratoriów. Jest ono napisane w języku Python3 i oparte o Projekt Jupyter. Wykorzystuje też biblioteki Numpy (który jest częścią biblioteki Scipy) oraz MatPlotLib.

4.1 Algorytm Fortune'a

4.1.1 Struktury danych

1. Klasa Point

   Klasa point reprezentuje punkt na płaszczyźnie dwuwymiarowej za pomocą współrzędnych x i y. Jednocześnie reprezentuje zdarzenia przechowywane w strukturze zdarzeń. Do tego celu potrzebne są pola:

   - orderingY - reprezentujące współrzędną sortującą zdarzenia,
   - arc – parabola, którą zaczyna dany punkt
   - edge -krawędź wieloboku Voronoi najbliżej danego punktu.

   Metody dostępne w tej klasie to:

   - __init__- Metoda inicjalizująca punkt,
   - setOrdering - Metoda ustawiająca współrzędną sortującą zdarzenia,
   - toPQ - Metoda zwracająca krotkę używaną do przetrzymywania w strukturze zdarzeń,
   - __hash__ - Metoda haszująca,
   - __eq__ - Metoda sprawdzająca identyczność,
   - __repr__ - Metoda zwracająca napis składający się z współrzędnych punktu.

2. Klasa HalfEdge

   Klasa HalfEdge reprezentuje półprostą lub odcinek na płaszczyźnie. Zawiera dwa pola: start, end reprezentujące początek i koniec odcinka, jeśli jedno z tych pól jest puste to mamy do czynienia z półprostą. Dodatkowymi polami są next oraz prev, dzięki nim możliwe jest utworzenie podwójnie łączonej listy krwędzi.
   Metody dostępne w tej klasie to:

   - __init__- Metoda inicjalizująca (początek i koniec na tym etapie nie istnieją),
   - __hash__ - Metoda haszująca,
   - __eq__ - Metoda sprawdzająca identyczność,
   - __repr__ - Metoda zwracająca napis składający się z punktu początkowego i końcowego.

3. Klasa RBNode

   Klasa RBNode reprezentuje węzeł w drzewie czerwono – czarnym, jest elementem struktury stanu. Jest używana do przedstawienia paraboli. Zawiera pola potrzebne do funkcjonowania drzewa czerwono czarnego: ojciec, lewy, prawy syn oraz kolor (parent, left, right, color). Pola reprezentujące parabolę to:

   - point – punkt, który inicjuje parabolę,
   - leftHalfEdge, rightHalfEdge – półproste przecinające parabolę,
   - prev, next – poprzednia oraz następna parabola,
   - triggeredBy – zdarzenie kołowe, które było przyczyną powstania paraboli.

4. Struktura stanu

   Struktura stanu reprezentowana jest przez drzewo czerwono czarne, w klasie RBTree. Klasa oferuje standardowe metody służące do używania drzewa czerwono czarnego, takie jak: sprawdzenie czy drzewo jest puste (isEmpty), ustawienie korzenia (createRoot), lewy obrót (left_rotate), prawy obrót(right_rotate), naprawę drzewa po dodaniu elementu (fix_insert), zamianę węzłów miejscami (transplant), usunięcie węzła (delete), naprawę drzewa po usunięciu (delete_fixup), znalezienie minimalnego węzła (minimum).

   Metody specyficzne da struktury stanu:

   - getNodeAbove – metoda zwracająca parabolę nad danym punktem,
   - insertBefore – metoda wstawiająca parabolę przed daną parabolą,
   - insertAfter – metoda wstawiająca parabolę po danej paraboli,
   - replace – metoda podmieniająca starą parabolę na nową (w tym samym miejscu w drzewie)

5. Struktura zdarzeń

   Struktura zdarzeń reprezentowana jest przez kolejkę priorytetową. Priorytet zdarzenia (punktu) jest definiowany przez pole orderingY.

6. Dodatkowe metody potrzebne do poprawnego działania algorytmu to:

   - getIntersectionOfParabolas – zwracająca punkt przecięcia się paraboli powstałych z dwóch podanych punktów, na danej współrzędnej y,
   - getConvergencePoint – zwracająca środek okręgu oraz dolny punkt okręgu powstałego z trzech podanych punktów

7. Klasa Voronoi

   Klasa Voronoi przechowuje oraz wyznacza diagram Voronoi dla podanego zbioru punktów.

   Pola zawierające się w tej klasie to:

   - points – zbiór punktów wejściowych,
   - events – struktura zdarzeń,
   - beachLine – struktura stanu,
   - notValidEvents – zbiór przetrzymujący nieprawidłowe zdarzenia kołowe,
   - vertices – zbiór punktów powstałego diagramu Voronoi,
   - listEdges – lista krawędzi diagramu Voronoi,
   - lowerLeft – lewy dolny punkt obramowania,
   - upperRight – prawy górny punkt obramowania.

   Główną metodą w tej klasie jest metoda „solve”, która po wywołaniu generuje diagram Voronoi.

   Główny podział metod to rozróżnienie na obsługujące zdarzenia kołowe oraz zdarzenia punktowe. Zdarzenia punktowe obsługiwane są przez metodę handleSiteEvent, wywołuje ona metody podrzędne:

   - breakArc – metoda dzieląca daną parabolę na trzy na danej współrzędnej y,
   - addCircleEvent – metoda dodająca zdarzenie kołowe powstałe z trzech parabol

   Zdarzenia kołowe obsługuje metoda handleCircleEvent, korzysta ona z następujących metod:

   - addCircleEvent (opisana wyżej)
   - removeArc – usuwa daną parabolę z struktury stanu,
     - addEdge – dodaje krawędź do listy krawędzi diagramu,

   Metody pomocnicze:

   - findBounds – metoda znajdująca punkty obramowania,
   - getIntersectionWithBox – metoda znajdująca punkt przecięcia półprostej z obramowaniem,
   - endHalfEdges – metoda kończąca wszystkie półproste (kończy w punkcie przecięcia z obramowaniem).

   Wyniki działania algorytmu reprezentowane są w klasie Voronoi przez:

   - pole vertices – zbiór wierzchołków diagramu,
   - pole listEdges – lista krawędzi

   Dodatkowo, dla każdego punktu ze zbioru wejściowego pole edge wskazuje na krawędź wieloboku otaczającego dany punkt, następnie dzięki polom next oraz prev krawędzi mamy dostęp do podwójnie łączonej listy krawędzi danego wieloboku.

4.1.2 Wizualizacja

Wizualizacja przeprowadzona za pomocą narzędzia Jupyter Notebook.

1. Klasa Visualization zajmuje się zapisywaniem do scen poszczególnych etapów działania algorytmu. Zawiera pola i metody odpowiedzialne za rysowanie poszczególnych elementów diagramu. Dla użytkownika najważniejszym polem jest pole scenes, zawierające wszystkie sceny wizualizacji.
2. Klasa VoronoiVisualization to klasa dziedzicząca z klasy Voronoi, nadpisuje część metod w celu dodania do nich funkcjonalności związanych z wizualizacją, nowa metoda to addVisualization dodająca obiekt klasy Visualization.

Aby poprawnie przeprowadzić wizualizację należy:

1. Zdefiniować zbiór obiektów klasy Point (set),
2. Stworzyć nowy obiekt klasy VoronoiVisualization, należy przekazać do konstruktora zbiór punktów oraz ustawić flagę steps na wartość True,
3. Stworzyć nowy obiekt klasy Visualization, do konstruktora przekazujemy wcześniej utworzony obiekt Voronoi,
4. Do obiektu Voronoi dodajemy wizualizację za pomocą metody addVisualization,
5. Wykonujemy metodę solve.
6. Definiujemy nowy obiekt klasy Plot, w konstruktorze przypisujemy do scenes pole scenes obiektu Visualization.
7. Na obiekcie plot wykonujemy metodę draw, z argumentem False.

4.1.3 Opis algorytmu

Algorytm Fortune'a opiera się na algorytmie zamiatania, działa w czasie O(n log n), oraz używa O(n) pamięci. Został zaprezentowany przez Stevena Fortune'a w 1986 roku. Miotłą w algorytmie jest pozioma linia poruszająca się w kierunku pionowym, z góry na dół. Struktura stanu nazywana jest linią brzegową. Składa się z łuków paraboli, z których każda jest wyznaczana przez konkretny punkt zbioru wejściowego. Punkty powyżej miotły zostały dołączone już do diagramu Voronoi, natomiast punkty poniżej nie zostały jeszcze rozważone.

4.2 Triangulacja Delaunay'a

Do obliczenia triangulacji użyty został algorytm Bowyera Watsona, następnie na jej postawie wyznaczone wieloboki Voronoi.

4.2.1 Reprezentacja

Poszczególne trójkąty reprezentowane są jako lista, zawierająca wierzchołki trójkąta, oraz wskazanie na sąsiednie trójkąty ($None$, jeżeli sąsiada nie ma).

4.2.2 Dostępne funkcje

1. Triangulacja
   - generate_points - generuje losową chmurę n punktów z zadanego przedziału
   - det - oblicza wyznacznik macierzy (pole równoległoboku rozpiętego na wektorach)
   - point_inside - sprawdza, czy punkt leży wewnątrz trójkąta, czy poza nim
   - equals - sprawdza, czy dane trójkąty są takie same
   - is_empty - sprawdza, czy dany trójkąt istnieje
   - triangle_center - zawraca środek ciężkości danego trójkąta
   - unit_vector - zwraca wektor jednostkowy (wersor), określający kierunek i zwrot wektora danego 2 punktami płaszczyzny
   - first_triangle - odnajduje pierwszy trójkąt aktualnej triangulacji
   - has_been_visited - sprawdza, czy dany trójkąt był juz odwiedzony
   - find_triangle - odszukuje trójkąt zawierający dany punkt, zaczynając od pierwszego trójkąta triangulacji, idąc po kolejnych sąsiadach
   - remove_triangle - całkowicie usuwa dany trójkąt, usuwając go również z listy sąsiadów dla każdego swojego sąsiada

- point_on_line - zwraca odpowiedni indeks, w zależności od tego, na którym boku trójkąta leży dany punkt, bądź $None$ jeżeli leży poza nim
- legal - sprawdza, czy krawędź jest legalna, tj. czy dany punkt leży na zewnątrz okręgu opisanego na danym trójkącie
- triangle_vertices - sprawdza, czy dane 2 punkty są wierzchołkami trójkąta
- get_ngh - zwraca sąsiada danego trójkąta względem krawędzi zadanej 2 punktami, $None$, jeśli takowego nie ma
- get_third_point - zwraca 3 punkt trójkąta, gdy dane są 2 pozostałe
- on_edge - sprawdza czy punkt leży na odcinku (danym 2 punktami)
- legalize - zamienia krawędź nielegalną na legalną (lokalnie Delaunay'a)
- change_ngh - zmienia odpowiedniego sąsiada danego trójkąta
- find_ngh_with_point - zwraca sąsiada trójkąta, takiego, który zawiera dany punkt
- insert_point - wstawia kolejny punkt do triangulacji
- triangulate - dokonuje triangulacji zadanej chmury punktów (wraz z 3 dodatkowymi punktami, tworzącymi tzw. *supertrójkąt*)
- triangle_out_of_scope - sprawdza, czy trójkąt jest poza zakresem współrzędnych; jeżeli jakikolwiek wierzchołek nie mieści się w zakresie, to cały trójkąt także
- get_result_triangles - usuwa z triangulacji trójkąty spoza zakresu, tj. zawierające punkty dodane sztucznie (*supertrójkąt*)
- **delaunay** - główna funkcja, dokonująca triangulacji zadanej chmury punktów płaszczyzny (poprzez dodanie *supertrójkąta*, triangulację zbioru z dodatkowymi punktami, usunięcie nadmiarowych trójkątów); zwraca trójkąty triangulacji

2. Diagram Voronoi

   - center_of_circumcircle - zwraca środek okręgu opisanego na danym trójkącie
   - triangles_to_lines - zwraca krawędzi danego trójkąta, na podstawie jego wierzchołków
   - find_edges_without_ngh - zwraca krawędzi danego trójkąta z podziałem na posiadające sasiadujący trójkąt oraz nieposiadające takowego
   - middle_point - zwraca punkt środkowy odcinka (zadanego jako 2 punkty do niego należące)
   - line_intersection - zwraca punkt przecięcia odcinków (zadanych jako 2 punkty do nich należące)
   - frame_cross - zwraca punkt przecięcia półprostej z ramką, daną skrajnymi punktami (lewy dolny i prawy górny róg)
   - **voronoi_diagram** - główna funkcja, dokonująca podziału płaszczyzny na wieloboki Voronoi dla zadanego zbioru punktów, za pomocą triangulacji Delaunay'a; zwraca wierzchołki oraz krawędzi diagramu Voronoi

4.2.3 Wizualizacja

W celu uzyskania wizualizacji algorytmów należy uruchomić funkcję **voronoi_diagram** z parametrem $visualize$ ustawionym na wartość $True$; zwrócone zostaną sceny odpowiednio dla triangulacji, jak i podziału na wieloboki Voronoi, które pozwalają na prześledzenie algorytmów krok po kroku.

Istnieje również możliwość zwizualizowania samego efektu działania algorytmu triangulacji bez kolejnych kroków. Aby to uzyskać należy uruchomić funkcję **delaunay** z parametrem $visualize$ ustawionym na wartość $False$, a na zwróconych przez nią trójkątach wywołać funkcję **triangles_to_lines**.

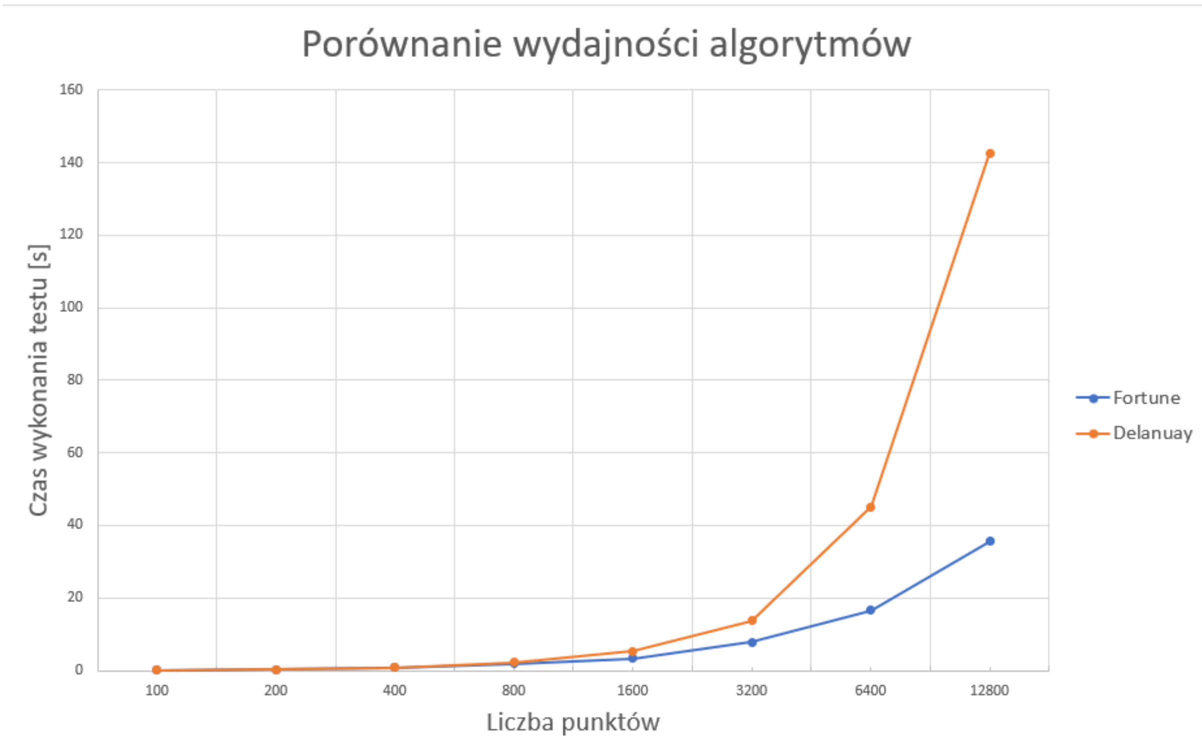
Podobnie można zwizualizować podział na wieloboki poprzez wywołanie funkcji **voronoi_diagram** z parametrem $visualize$ ustawionym na wartość $True$, a na podstawie zwróconych wierzchołków i krawędzi utworzyć wykres.

4.2.4 Opis algorytmu

Algorytm Bowyer'a - Watsona pochodzi z roku 1981, polega na skonstruowaniu powłoki pokrywającej punkty chmury poprzez dodanie 3 punktów pomocniczych tworząc trójkąt, który zawiera w swoim wnętrzu wszystkie punkty chmury. Jest to początkowa triangulacja. Następnie dokłada się kolejne punkty (w kolejności losowej), odnajdując trójkąt triangulacji, który go zawiera. W zależności od tego, czy punkt leży wewnątrz, czy na krawędzi trójkąta, odpowiednio dzieli się trójkąty, następnie następuje legalizacja krawędzi w nowo-powstałych trójkątach. Finalnie otrzymana triangulacja nie zawiera dodanych na początku 3 punktów.
Korzystając z faktu, że triangulacja Delaunay'a jest grafem dualnym diagramu Voronoi, w stosunkowo łatwy sposób da się otrzymać podział płaszczyzny na wieloboki, poprzez połączenie środków okręgów opisanych na sąsiadujących ze sobą (uprzednio wyznaczonych) trójkątach. Jeżeli natomiast dany trójkąt nie ma sąsiada względem którejś z krawędzi, utworzona zostaje półprosta o początku w środku okręgu, której przedłużenie jest symetralną boku boku trójkąta, względem którego nie ma sąsiada.

5 Przeprowadzone testy

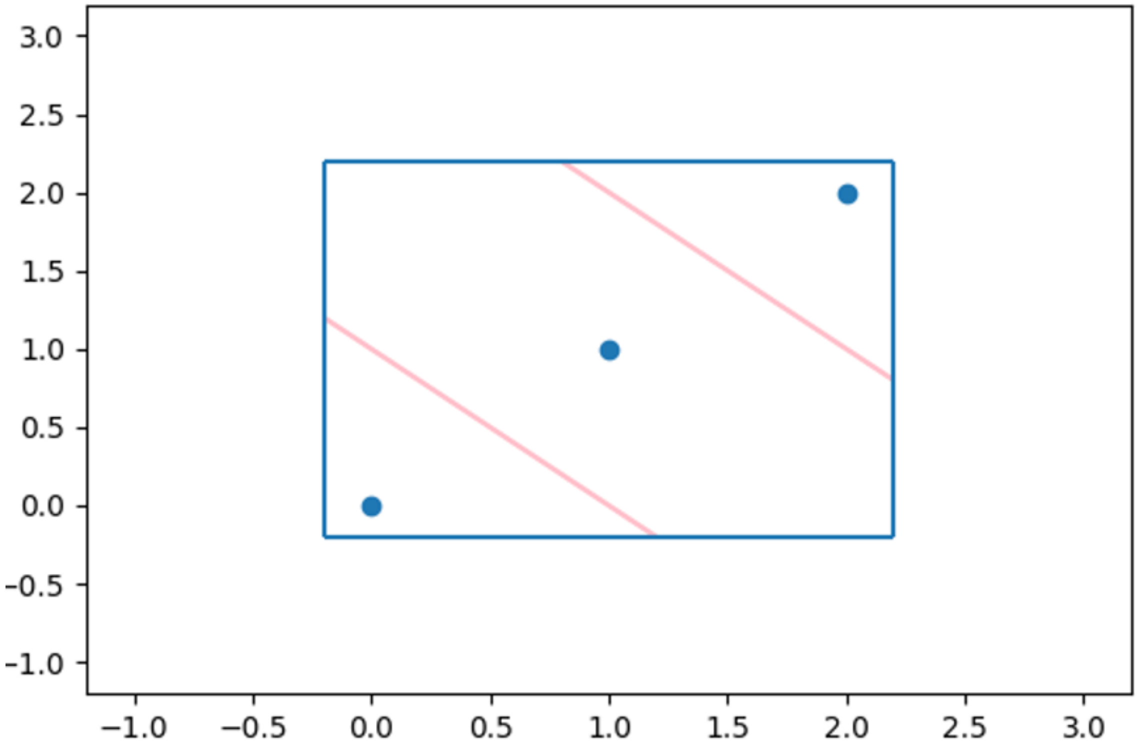
Algorytmy działają poprawnie dla zbiorów danych większych od trzech punktów, jednocześnie nieposiadających punktów mających te same współrzędne x lub y.
Testy wydajnościowe zostały przeprowadzone na losowych zestawach danych, dla każdej ilości punktów zostały powtórzone osiem razy, aby uniknąć wpływu skrajnych przypadków na czas działania. Uśrednione wyniki zostały przedstawione graficznie na Rysunku 1.



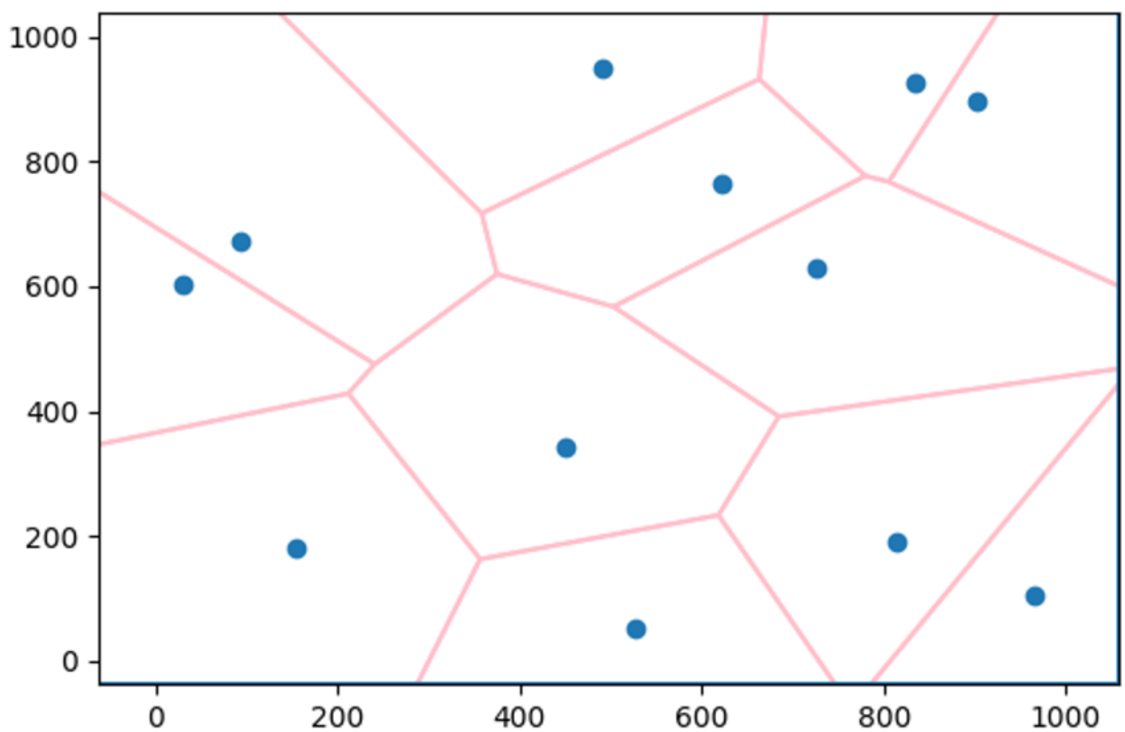
Rysunek 1: Porównanie czasów działania obu użytych algorytmów

6 Przykładowe wyniki działania

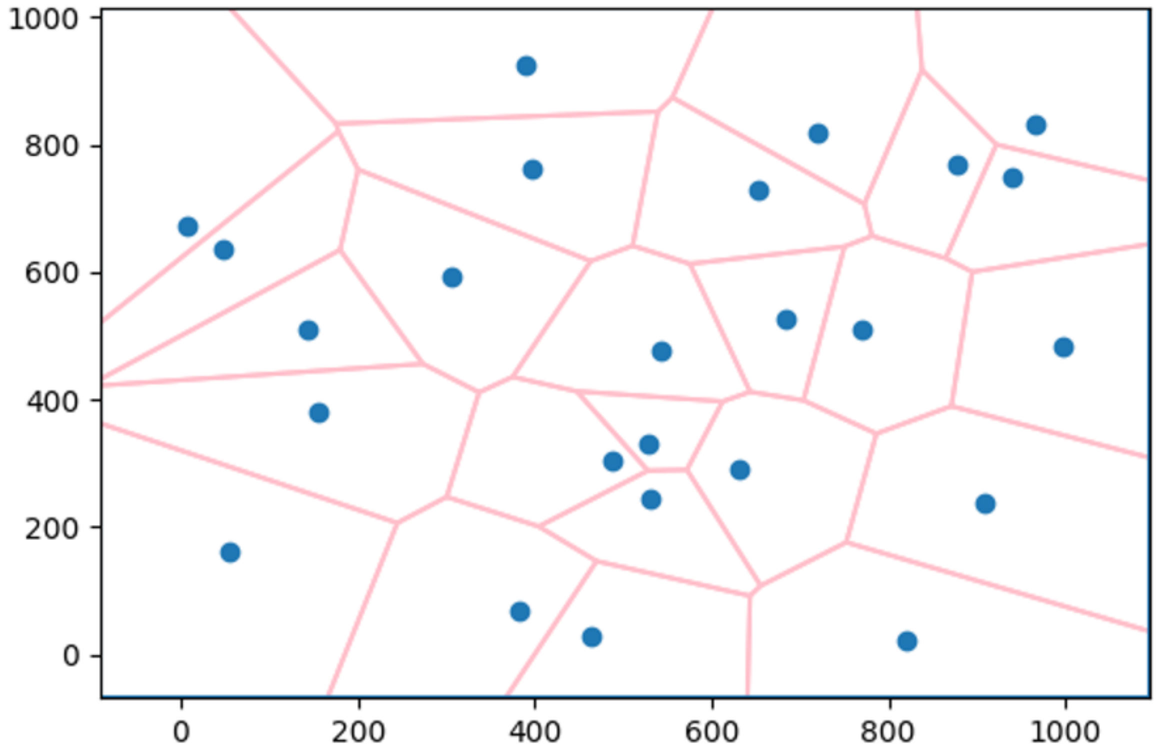
6.1 Algorytm Fortune'a



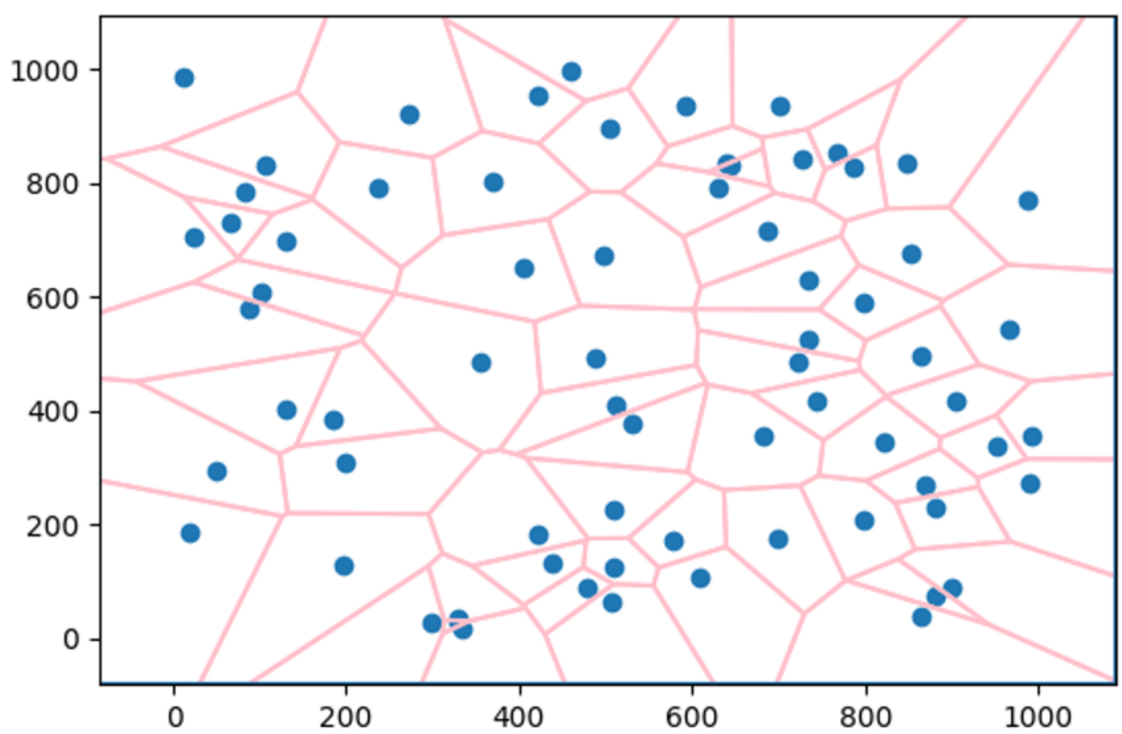
Rysunek 2: Zestaw 3 punktów leżących na prostej



Rysunek 3: Zestaw 8 losowych punktów

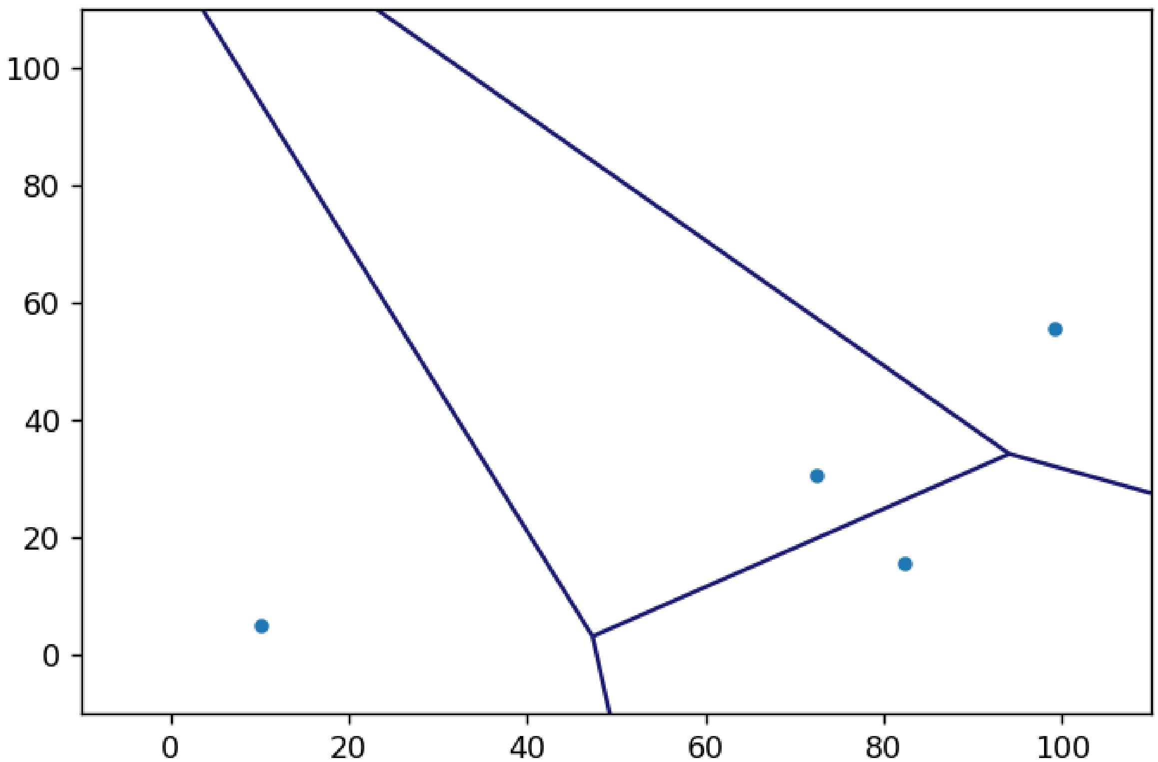


Rysunek 4: Zestaw 25 losowych punktów

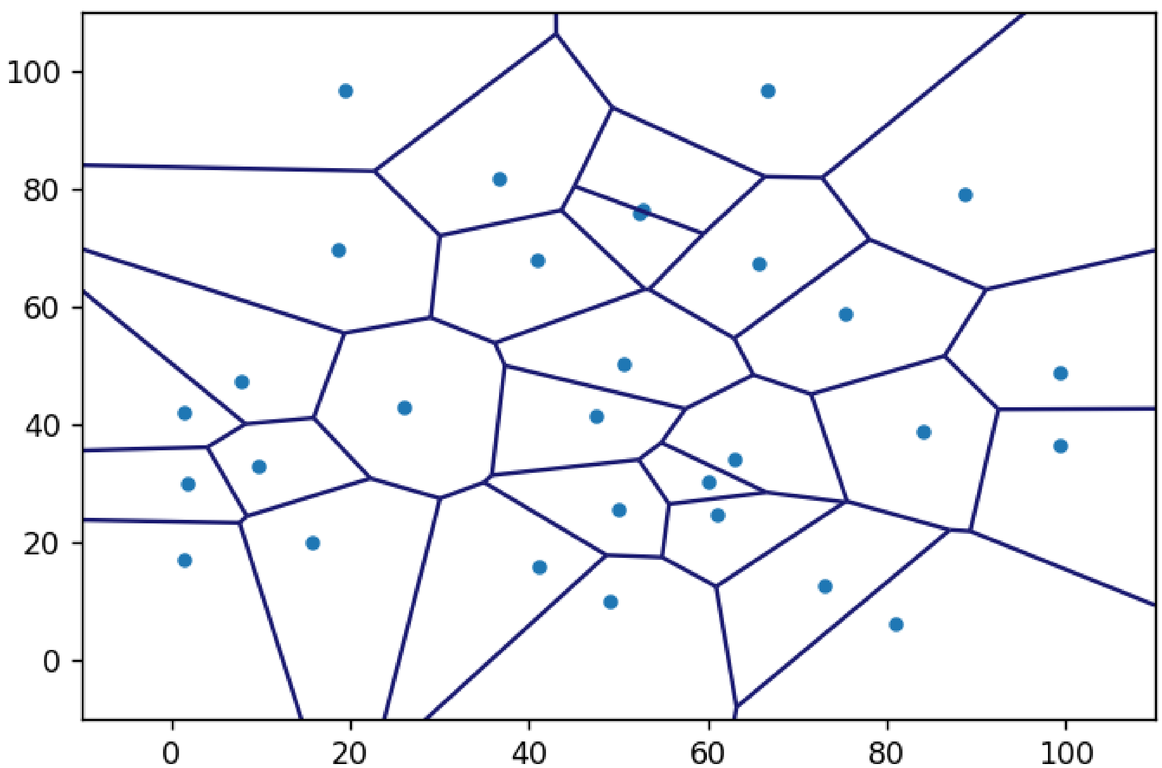


Rysunek 5: Zestaw 70 losowych punktów

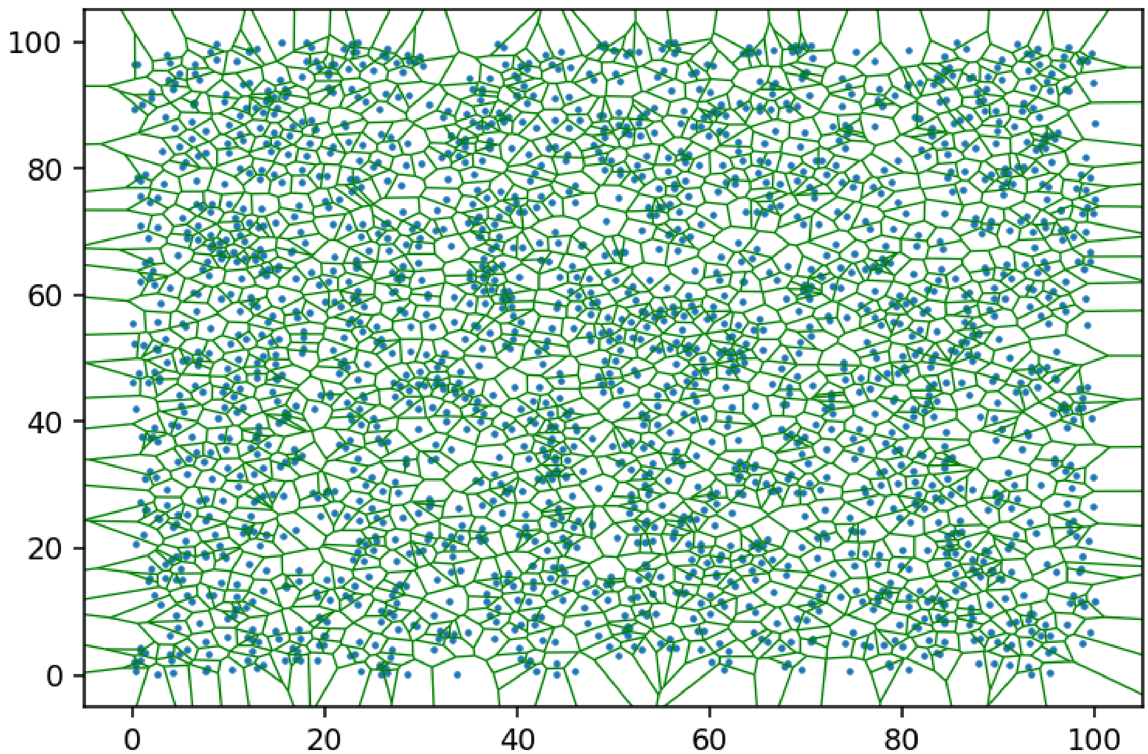
6.2 Algorytm na podstawie triangulacji Delaunay'a



Rysunek 6: Zestaw 4 losowych punktów

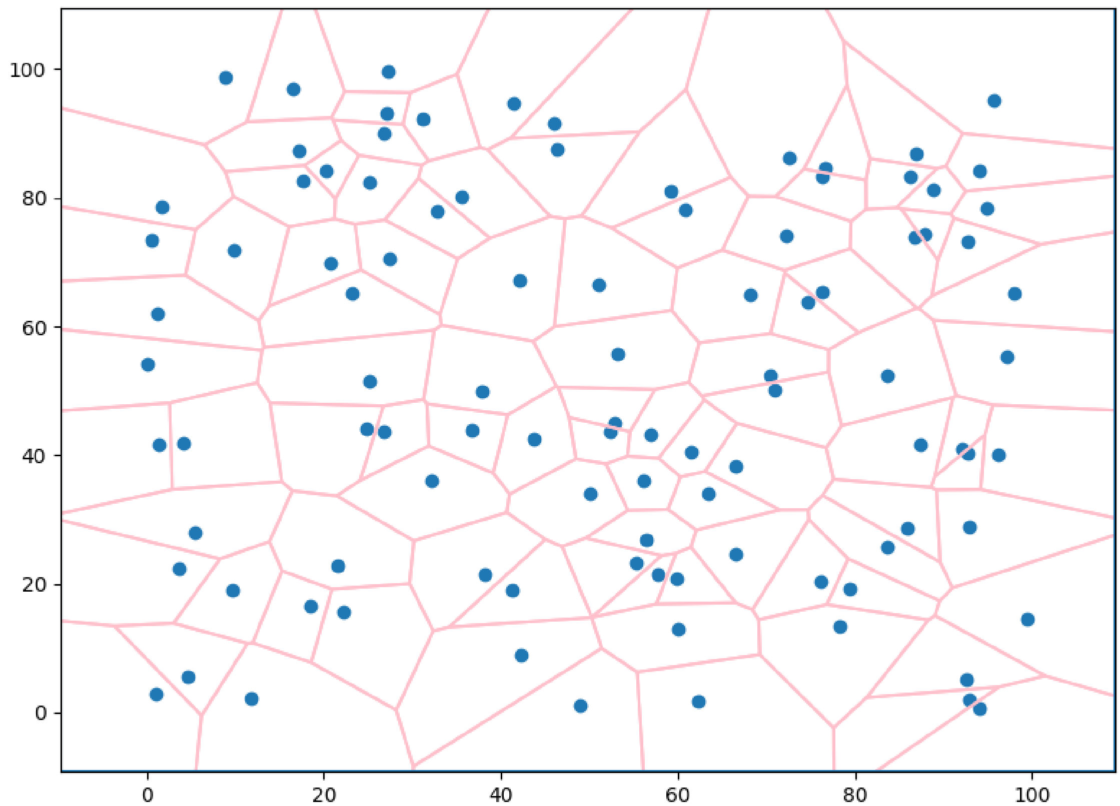


Rysunek 7: Zestaw 30 losowych punktów

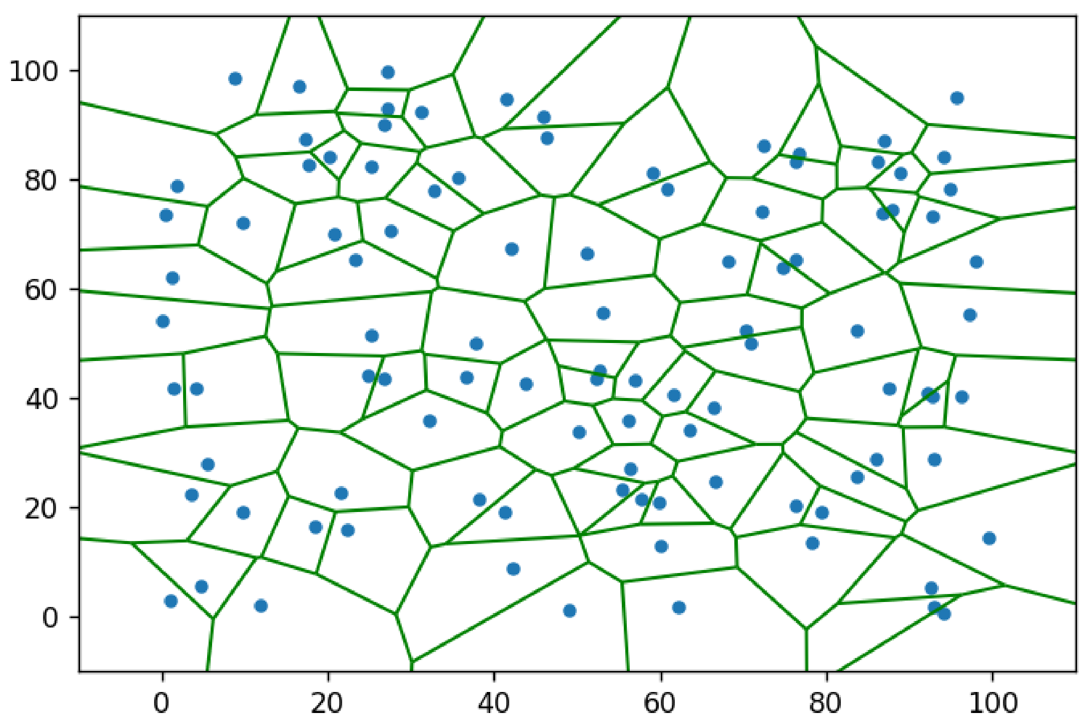


Rysunek 8: Zestaw 2000 losowych punktów

6.3 Oba algorytmy na tym samym losowym zestawie danych



Rysunek 9: Zestaw 100 losowych punktów - algorytm Fortune'a



Rysunek 10: Zestaw 100 losowych punktów - algorytm na podstawie triangulacji Delaunay'a

7 Wnioski

Oba testowane algorytmy wyznaczają wieloboki Voronoi dla zadanych punktów poprawnie, pod warunkiem, że zbiory są większe od 3 punktów oraz nie leżą one na tej samej prostej. Zgodnie z przewidywaniami algorytm Fortune'a wykazuje czasową przewagę nad algorytmem na podstawie triangulacji Delaunay'a (metoda Bowyer'a Watsona).